Python CheatSheet

Basics

Basic syntax from the python programming language

Showing Output To User

The print function is used to display or print output as follows

```
print("Content that you wanna print on screen")
```

we can display the content present in object using prit function as follows:-

```
var1 = "Shruti"
print("Hi my name is: ",var1)
```

Taking Input From the User

The input function is used to take input as string or character from the user as follows:

```
var1 = input("Enter your name: ")
print("My name is: ", var1)
```

To take input in form of other datatypes we need to typecaste them as follows:-
To take input as an integer:-

```
var1=int(input("enter the integer value"))
print(var1)
```

To take input as an float:-

```
var1=float(input("enter the float value"))
print(var1)
```

range Function

range function returns a sequence of numbers, eg, numbers starting from 0 to n-1 for range(0, n)

```
range(int_start_value,int_stop_value,int_step_value)
```

Here the start value and step value are by default 1 if not mentioned by the programmer.
but int_stop_value is the compulsory parameter in range function
example-
Display all even numbers between 1 to 100

```
for i in range(0,101,2):
        print(i)
```

Comments

Comments are used to make the code more understandable for programmers, and they are not executed by compiler or interpreter.

Single line comment

```
# This is a single line comment
```

Multi-line comment

```
'''This is a
multi-line
comment'''
```

Escape Sequence

An escape sequence is a sequence of characters; it doesn't represent itself (but is translated into another character) when used inside string literal or character. Some of the escape sequence characters are as follows:

Newline

Newline Character

```
print("\n")
```

Backslash

It adds a backslash

```
print("\\")
```

Single Quote

It adds a single quotation mark

```
print("\'")
```

Tab

It gives a tab space

```
print("\t")
```

Backspace

It adds a backspace

```
print("\b")
```

Octal value

It represents the value of an octal number

```
print("\ooo")
```

Hex value

It represents the value of a hex number

```
print("\xhh")
```

Carriage Return

Carriage return or \r will just work as you have shifted your cursor to the beginning of the string or line.

```
pint("\r")
```

Strings

Python string is a sequence of characters, and each character can be individually accessed using its index.

String

You can create Strings by enclosing text in both forms of quotes - single quotes or double quotes.

```
variable_name = "String Data"
```

example

```
str="Shruti"
print("string is ",str)
```

Indexing

The position of every character placed in the string starts from 0th position ans step by step it ends at length-1 position

Slicing

Slicing refers to obtaining a sub-string from the given string. The following code will include index 1, 2, 3, and 4 for the variable named **var_name**
Slicing of the string can be obtained by the following syntax-

```
string_var[int_start_value:int_stop_value:int_step_value]
```

```
var_name[1 : 5]
```

here start and step value are considered 0 and 1 respectively if not mentioned by the programmmer

isalnum() method

Returns True if all the characters in the string are alphanumeric, else False

```
string_variable.isalnum()
```

isalpha() method

Returns True if all the characters in the string are alphabets

```
string_variable.isalpha()
```

isdecimal() method

Returns True if all the characters in the string are decimals

```
string_variable.isdecimal()
```

isdigit() method

Returns True if all the characters in the string are digits

```
string_variable.isdigit()
```

islower() method

Returns True if all characters in the string are lower case

```
string_variable.islower()
```

isspace() method

Returns True if all characters in the string are whitespaces

```
string_variable.isspace()
```

isupper() method

Returns True if all characters in the string are upper case

```
string_variable.isupper()
```

lower() method

Converts a string into lower case equivalent

```
string_variable.lower()
```

upper() method

Converts a string into upper case equivalent

```
string_variable.upper()
```

strip() method

It removes leading and trailing spaces in the string

```
string_variable.strip()
```

List

A List in Python represents a list of comma-separated values of any data type between square brackets.

```
var_name = [element1, element2, ...]
```

These elements can be of different datatypes

Indexing

The position of every elements placed in the string starts from 0th position ans step by step it ends at length-1 position
List is ordered,indexed,mutable and most flexible and dynamic collection of elements in python.

Empty List

This method allows you to create an empty list

```
my_list = []
```

index method

Returns the index of the first element with the specified value

```
list.index(element)
```

append method

Adds an element at the end of the list

```
list.append(element)
```

extend method

Add the elements of a given list (or any iterable) to the end of the current list

```
list.extend(iterable)
```

insert method

Adds an element at the specified position

```
list.insert(position, element)
```

pop method

Removes the element at the specified position and returns it

```
list.pop(position)
```

remove method

The remove() method removes the first occurrence of a given item from the list

```
list.remove(element)
```

clear method

Removes all the elements from the list

```
list.clear()
```

count method

Returns the number of elements with the specified value

```
list.count(value)
```

reverse method

Reverses the order of the list

```
list.reverse()
```

sort method

Sorts the list

```
list.sort(reverse=True|False)
```

Tuples

Tuples are represented as comma-separated values of any data type within parentheses.

Tuple Creation

```
variable_name = (element1, element2, ...)
```

These elements can be of different datatypes

Indexing

The position of every elements placed in the string starts from 0th position ans step by step it ends at length-1 position
Tuples are ordered,indexing,immutable and most secured collection of elements
Lets talk about some of the tuple methods:

count method

It returns the number of times a specified value occurs in a tuple

```
tuple.count(value)
```

index method

It searches the tuple for a specified value and returns the position.

```
tuple.index(value)
```

Sets

A set is a collection of multiple values which is both unordered and unindexed. It is written in curly brackets.

Set Creation: Way 1

```
var_name = {element1, element2, ...}
```

Set Creation: Way 2

```
var_name = set([element1, element2, ...])
```

Set is unordered,immutable,non-indexed type of collection.Duplicate elements are not allowed in sets.

Set Methods

Lets talk about some of the methods of sets:

add() method

Adds an element to a set

```
set.add(element)
```

clear() method

Remove all elements from a set

```
set.clear()
```

discard() method

Removes the specified item from the set

```
set.discard(value)
```

intersection() method

Returns intersection of two or more sets

```
set.intersection(set1, set2 ... etc)
```

issubset() method

Checks if a set is a subset of another set

```
set.issubset(set)
```

pop() method

Removes an element from the set

```
set.pop()
```

remove() method

Removes the specified element from the set

```
set.remove(item)
```

union() method

Returns the union of two or more sets

```
set.union(set1, set2...)
```

Dictionaries

The dictionary is an unordered set of comma-separated key:value pairs, within {}, with the requirement that within a dictionary, no two keys can be the same.

Dictionary

```
<dictionary-name> = {<key>: value, <key>: value ...}
```

Dictionary is ordered and mutable collection of elements.Dictionary allows duplicate values but not duplicate keys.

Empty Dictionary

By putting two curly braces, you can create a blank dictionary

```
mydict={}
```

Adding Element to a dictionary

By this method, one can add new elements to the dictionary

```
<dictionary>[<key>] = <value>
```

Updating Element in a dictionary

If a specified key already exists, then its value will get updated

```
<dictionary>[<key>] = <value>
```

Deleting an element from a dictionary

del keyword is used to delete a specified key:value pair from the dictionary as follows:

```
del <dictionary>[<key>]
```

Dictionary Functions & Methods

Below are some of the methods of dictionaries

len() method

It returns the length of the dictionary, i.e., the count of elements (key: value pairs) in the dictionary

```
len(dictionary)
```

clear() method

Removes all the elements from the dictionary

```
dictionary.clear()
```

get() method

Returns the value of the specified key

```
dictionary.get(keyname)
```

items() method

Returns a list containing a tuple for each key-value pair

```
dictionary.items()
```

keys() method

Returns a list containing the dictionary's keys

```
dictionary.keys()
```

values() method

Returns a list of all the values in the dictionary

```
dictionary.values()
```

update() method

Updates the dictionary with the specified key-value pairs

```
dictionary.update(iterable)
```

Indentation

In Python, indentation means the code is written with some spaces or tabs into many different blocks of code to indent it so that the interpreter can easily execute the Python code.
Indentation is applied on conditional statements and loop control statements. Indent specifies the block of code that is to be executed depending on the conditions.

Conditional Statements

The if, elif and else statements are the conditional statements in Python, and these implement selection constructs (decision constructs).

if Statement

```
if(conditional expression):
    statements
```

if-else Statement

```
if(conditional expression):
    statements
else:
    statements
```

if-elif Statement

```
if (conditional expression):
    statements
elif (conditional expression):
    statements
else:
    statements
```

Nested if-else Statement

```
if (conditional expression):
    if (conditional expression):
        statements
    else:
        statements
else:
    statements
```

example-

```
a=15
b=20
c=12
if(a>b and a>c):
      print(a,"is greatest")
elif(b>c and b>a):
      print(b," is greatest")
else:
    print(c,"is greatest")
```

Loops in Python

A loop or iteration statement repeatedly executes a statement, known as the loop body, until the controlling expression is false (0).

For Loop

The for loop of Python is designed to process the items of any sequence, such as a list or a string, one by one.

```
for <variable> in <sequence>:
    statements_to_repeat
```

example-

```
for i in range(1,101,1):
           print(i)
```

While Loop

A while loop is a conditional loop that will repeat the instructions within itself as long as a conditional remains true.

```
while <logical-expression>:
    loop-body
```

example-

```
i=1
while(i<=100):
        print(i)
        i=i+1
```

Break Statement

The break statement enables a program to skip over a part of the code. A break statement terminates the very loop it lies within.

```
for <var> in <sequence>:
    statement1
    if <condition>:
        break
    statement2
statement_after_loop
```

example-

```
for i in range(1,101,1):
    print(i ,end=" ")
```

```
    if(i==50):
        break
    else:
        print("Mississippi")
print("Thank you")
```

Continue Statement

The continue statement skips the rest of the loop statements and causes the next iteration to occur.

```
for <var> in <sequence>:
    statement1
    if <condition> :
        continue
    statement2
    statement3
    statement4
```

example-

```
for i in [2,3,4,6,8,0]:
    if (i%2!=0):
        continue
    print(i)
```

Functions

A function is a block of code that performs a specific task. You can pass parameters into a function. It helps us to make our code more organized and manageable.

Function Definition

```
def my_function():
          #statements
```

def keyword is used before defining the function.

Function Call

```
my_function()
```

Whenever we need that block of code in our program simply call that function name whenever neeeded. If parameters are passed during defing the function we have to pass the parameters while calling that function
example-

```
def add():       #function defination
        a=10
        b=20
       print(a+b)
add()            #function call
```

Return statement in Python function

The function return statement return the specified value or data item to the caller.

```
return [value/expression]
```

Arguments in python function

Arguments are the values passed inside the parenthesis of the function while defining as well as while calling.

```
def my_function(arg1,arg2,arg3....argn):
          #statements
my_function(arg1,arg2,arg3....argn)
```

example-

```
def add(a,b):
      return a+b
x=add(7,8)
print(x)
```

File Handling

File handling refers to reading or writing data from files. Python provides some functions that allow us to manipulate data in the files.

open() function

```
var_name = open("file name", " mode")
```

modes-

1. **r** - to read the content from file
2. **w** - to write the content into file
3. **a** - to append the existing content into file
4. **r+:** To read and write data into the file. The previous data in the file will be overridden.
5. **w+:** To write and read data. It will override existing data.
6. **a+:** To append and read data from the file. It won't override existing data.

close() function

```
var_name.close()
```

read () function

The read functions contains different methods, read(),readline() and readlines()

```
read() #return one big string
```

It returns a list of lines

```
readlines() #returns a list
```

It returns one line at a time

```
readline #returns one line at a time
```

write function

This function writes a sequence of strings to the file.

```
write() #Used to write a fixed sequence of characters to a file
```

It is used to write a list of strings

```
writelines()
```

Exception Handling

An exception is an unusual condition that results in an interruption in the flow of a program.

try and except

A basic try-catch block in python. When the try block throws an error, the control goes to the except block.

```
try:
    [Statement body block]
    raise Exception()
except Exceptionname:
    [Error processing block]
```

else

The else block is executed if the try block have not raise any exception and code had been running successfully

```
try:
     #statements
except:
     #statements
else:
      #statements
```

finally

Finally block will be executed even if try block of code has been running successsfully or except block of code is been executed. finally block of code will be executed compulsory

Object Oriented Programming (OOPS)

It is a programming approach that primarily focuses on using objects and classes. The objects can be any real-world entities.

class

The syntax for writing a class in python

```
class class_name:
    pass #statements
```

Creating an object

Instantiating an object can be done as follows:

```
<object-name> = <class-name>(<arguments>)
```

self parameter

The self parameter is the first parameter of any function present in the class. It can be of different name but this parameter is must while defining any function into class as it is used to access other data members of the class

class with a constructor

Constructor is the special function of the class which is used to initialize the objects. The syntax for writing a class with the constructor in python

```
class CodeWithHarry:

    # Default constructor
    def __init__(self):
        self.name = "CodeWithHarry"

    # A method for printing data members
    def print_me(self):
        print(self.name)
```

Inheritance in python

By using inheritance, we can create a class which uses all the properties and behavior of another class. The new class is known as a derived class or child class, and the one whose properties are acquired is known as a base class or parent class.
It provides the re-usability of the code.

```
class Base_class:
       pass
class derived_class(Base_class):
        pass
```

Types of inheritance-

- Single inheritance
- Multiple inheritance
- Multilevel inheritance
- Hierarchical inheritance

filter function

The filter function allows you to process an iterable and extract those items that satisfy a given condition

```
filter(function, iterable)
```

issubclass function

Used to find whether a class is a subclass of a given class or not as follows

```
issubclass(obj, classinfo) # returns true if obj is a subclass of classinfo
```

Iterators and Generators

Here are some of the advanced topics of the Python programming language like iterators and generators

Iterator

Used to create an iterator over an iterable

```
iter_list = iter(['Harry', 'Aakash', 'Rohan'])
print(next(iter_list))
print(next(iter_list))
print(next(iter_list))
```

Generator

Used to generate values on the fly

```
# A simple generator function
def my_gen():
    n = 1
    print('This is printed first')
    # Generator function contains yield statements
    yield n
    n += 1
    print('This is printed second')
    yield n
    n += 1
    print('This is printed at last')
    yield n
```

Decorators

Decorators are used to modifying the behavior of a function or a class. They are usually called before the definition of a function you want to decorate.

property Decorator (getter)

```
@property
def name(self):
    return self.__name
```

setter Decorator

It is used to set the property 'name'

```
@name.setter
def name(self, value):
    self.__name=value
```

deleter Decorator

It is used to delete the property 'name'

```
@name.deleter #property-name.deleter decorator
def name(self, value):
    print('Deleting..')
    del self.__name
```